Overlap2D

INTRODUCTION

What is Overlap2D?

Overlap2D is an open source level map and UI editor for game development. It consists of an interface based in WYSIWYG with a canvas surrounded by a variety of tools to work in real time.

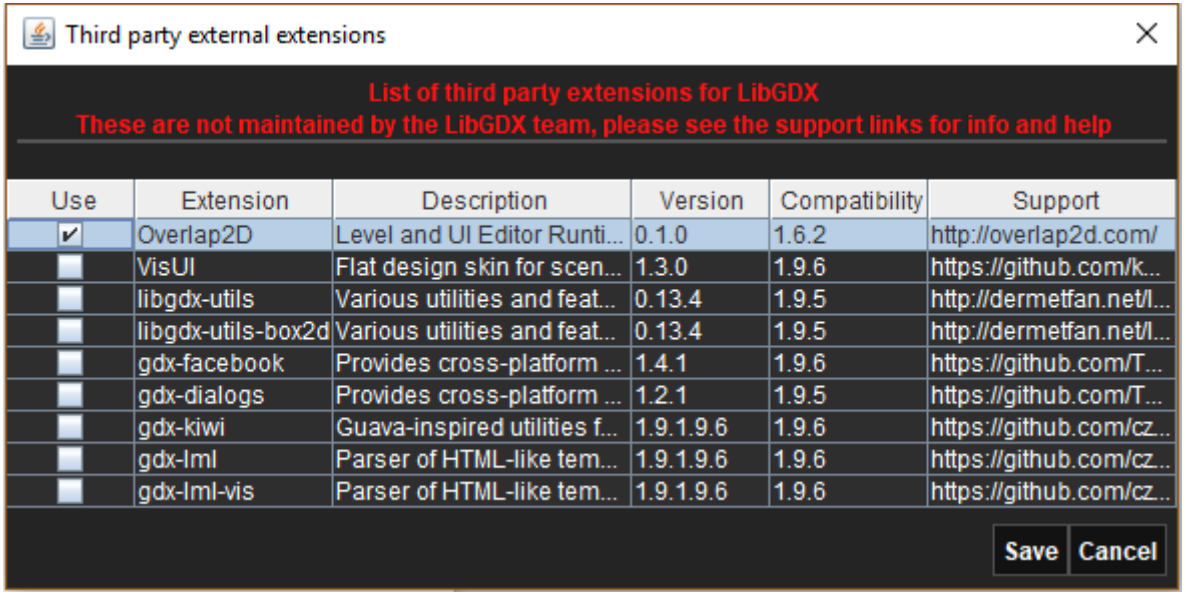
In this presentation we will explain how to integrate it with LibGDX.

SETTING UP

Runtime

The fastest and easiest way is to use the built-in Overlap2D runtime within the LibGDX Project Generator.

We can find it inside “Show Third Party Extensions” menu.



SETTING UP

The Editor

The Overlap2D editor comes in the form of an executable .jar we need to download from the website*.

Starting to work with Overlap2D is as easy as double clicking to launch the editor and we are all set up and ready to create.

*At this time, the website if down due to a recent attack, but you can alternatively download it from their GitHub page https://github.com/UnderwaterApps/overlap2d.

SETTING UP

Final Tweaks

Before we can start working with both LibGDX and Overlap2D we need to edit our gradle settings if we don't want to encounter errors in our programs.

LibGDX version

We are required to downgrade the LibGDX version to 1.9.4 if we want to work with animations or else the application will crash.

```
ext {
        appName = "my-gdx-game"
        gdxVersion = '1.9.4'
        roboVMVersion = '2.3.1'
        box2DLightsVersion = '1.4'
        ashleyVersion = '1.7.0'
        aiVersion = '1.8.0'
    }
```

SETTING UP

Runtime version

The default version of the runtime dependency created by our LibGDX Project Generator is heavily outdated and we need to modify it to reflect the latest version (0.1.2-SNAPSHOT at this moment)

```
project(":core") {
    apply plugin: "java"


    dependencies {
        compile "com.badlogicgames.gdx:gdx:$gdxVersion"
        compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
        compile "com.underwaterapps.overlap2druntime:overlap2d-runtime-libgdx:0.1.2-SNAPSHOT"
    }
}
```

THE EDITOR

Main Overlap2D features

- Drag and Drop System
- Layers
- Toolboxes for real-time editing

Exporting the Project

After we are done creating and editing our project, we need to export it to integrate it with our LibGDX project. All we need to do is go to Export Settings, choose the destination (our LibGDX assets folder in our case) and export it. Whenever we make any changes to the project, we need to export it again to reflect these.

THE CODE

Integration with LibGDX

Now that everything is in place, we need to add the logic to give new life to our world. Another powerful feature of the Overlap2D engine is that many things like movement or transformations are managed internally, so the code and logic needed is heavily reduced. Our code is going to use the next components:

- SceneLoader class
- ItemWrapper class
- IScript interface

THE CODE

SceneLoader

As its name suggests, it is the class in control of the scene management. It is used to load a selected scene and render it on the screen.

First we need to initialize it, preferably with a viewport.

```
viewport = new FitViewport(1280, 720);
sceneLoader = new SceneLoader();
sceneLoader.loadScene("MainScene", viewport);
```

We need to update its status in every rendered frame.

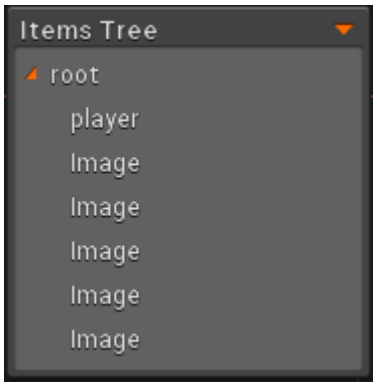
```
@Override
    public void render() {
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        sceneLoader.getEngine().update(Gdx.graphics.getDeltaTime());
    }
```

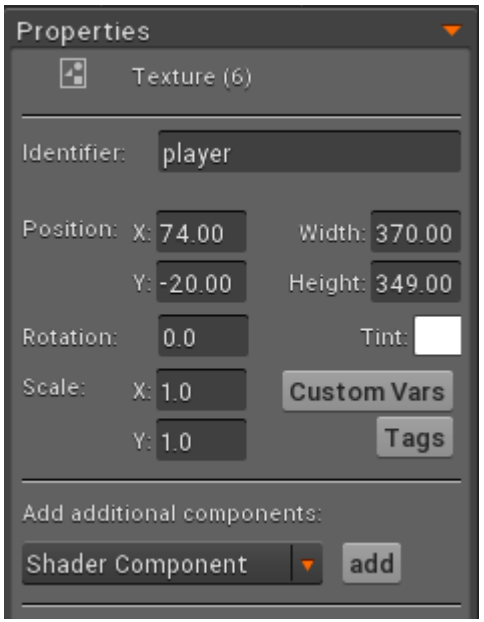
THE CODE

ItemWrapper

The ItemWrapper class holds the assets inside our Items Tree of our project, so if we need to access to any of its resources, we use this class to pick it up. It is important to use the identifiers, for it is the most optimal way to direct the ItemWrapper to the item we need to get from it.

To get an item called player from our assets, first we make sure we got it correctly identified.





THE CODE

Once we have our resource aproppriately tagged, all we need to do is initialize the ItemWrapper and tell it to get it for us.

```
ItemWrapper root = new ItemWrapper(sceneLoader.getRoot());
root.getChild("player")
```

The parameter in the contructor is the root from our scene, so we need to point the ItemWrapper to the scene we want it to retrieve the items from.

THE CODE

IScript Interface

This interface works much like an Actor class and it's the one in charge of scripting the behaviour of our world components. Its structure is as follows:

```
@Override
public void init(Entity entity) {

}

@Override
public void act(float f) {

}

@Override
public void dispose() {

}
```

THE CODE

We use init to initialize its status, act to implement the logic of our scripted behaviour and dispose is self explanatory.

Once we have created a class implementing this interface, we need to apply it to the assets that are going to make use of it by retrieving it with our ItemWrapper and then assigning it our Iscript.

```
playerIS = new PlayerIScript(0f);
ItemWrapper root = new ItemWrapper(sceneLoader.getRoot());
root.getChild("player").addScript(playerIS);
```

LIGHTING

Working with lights

One of the sweetest features the Overlap2D comes with is the ease of creating lighting effects. All we need to do is enable it in our project, and drag and drop the light tool.

They come in two flavors: cone light, and round light. Cone light emites light in a cone, and you can manipulate the angle, size and direction of the cone. Round light is exactly the same, but the light irradiates in every direction in a radius around the source, and you can also modify its length, radius, etc…

Lights are handled in the same way as images or animations are, so you can make them move or script its behaviour as we do with the rest of the assets.

PLUGINS

NinePatch converter

Overlap2D comes with several plugins, but an honorable mention goes to the NinePatch converter, which makes it very easy and very conformatable to transform a standard image to a NinePatch one.

All we need to do is left-click our image, select the Convert to NinePatch option in the context menu, and edit our image as we see fit. Once we hit the Apply button we have our new image ready to go.